



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/646,309	08/22/2003	Gregory M. Wright	SUN-P9042	9198
57960 7590 11/12/2009 PVF -- SUN MICROSYSTEMS INC. C/O PARK, VAUGHAN & FLEMING LLP 2820 FIFTH STREET DAVIS, CA 95618-7759			EXAMINER CHEN, QING	
			ART UNIT 2191	PAPER NUMBER
			MAIL DATE 11/12/2009	DELIVERY MODE PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary

Application No.

10/646,309

Applicant(s)

WRIGHT ET AL.

Examiner

Qing Chen

Art Unit

2191

Period for Reply -- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

- 1) ☒ Responsive to communication(s) filed on 22 July 2009.
- 2a) ☒ This action is **FINAL**. 2b) ☐ This action is non-final.
- 3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

- 4) ☒ Claim(s) 1, 2, 4-11, 13-18 and 28-35 is/are pending in the application.
- 4a) Of the above claim(s) _____ is/are withdrawn from consideration.
- 5) ☐ Claim(s) _____ is/are allowed.
- 6) ☒ Claim(s) 1, 2, 4-11, 13-18 and 28-35 is/are rejected.
- 7) ☐ Claim(s) _____ is/are objected to.
- 8) ☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

- 9) ☐ The specification is objected to by the Examiner.
- 10) ☐ The drawing(s) filed on _____ is/are: a) ☐ accepted or b) ☐ objected to by the Examiner.
Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
- 11) ☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

- 12) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
- a) ☐ All b) ☐ Some * c) ☐ None of:
1. ☐ Certified copies of the priority documents have been received.
 2. ☐ Certified copies of the priority documents have been received in Application No. _____.
 3. ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

- 1) ☐ Notice of References Cited (PTO-892)
- 2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
- 3) ☐ Information Disclosure Statement(s) (PTO/SI/08)
Paper No(s)/Mail Date _____
- 4) ☐ Interview Summary (PTO-413)
Paper No(s)/Mail Date _____
- 5) ☐ Notice of Informal Patent Application
- 6) ☐ Other: _____

DETAILED ACTION

1. This Office action is in response to the amendment filed on July 22, 2009.
2. **Claims 1, 2, 4-11, 13-18, and 28-35** are pending.
3. **Claims 1, 2, 4-6, 9-11, 13-15, 18, and 28-35** have been amended.
4. **Claims 3, 12, 19-27, and 36-39** have been canceled.
5. The objections to Claims 1, 4-8, 10, 13-17, and 28-35 are withdrawn in view of Applicant's amendments to the claims. However, Applicant's amendments to the claims fail to fully address the objections to Claims 2, 9, 11, and 18 due to improper antecedent basis. Accordingly, these objections are maintained and further explained hereinafter.
6. The 35 U.S.C. § 112, second paragraph, rejections of Claims 28-31 are withdrawn in view of Applicant's amendments to the claims. However, Applicant's amendments to the claims fail to address the rejections to Claims 32-35 due to insufficient antecedent basis. Accordingly, these rejections are maintained and further explained hereinafter.

Response to Amendment

Claim Objections

7. **Claims 2, 9, 11, and 18** are objected to because of the following informalities:
 - **Claims 2, 9, 11, and 18** recite the limitation "the call to the selected native code method." Applicant is advised to change this limitation to read "the call to any native code method" for the purpose of providing it with proper explicit antecedent basis. Appropriate correction is required.

Claim Rejections - 35 USC § 112

8. The following is a quotation of the second paragraph of 35 U.S.C. 112:

The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

9. **Claims 32-35** are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

Claim 32 recites the limitation “the combined intermediate representation.” There is insufficient antecedent basis for this limitation in the claim. In the interest of compact prosecution, the Examiner subsequently interprets this limitation as reading “the integrated intermediate representation” for the purpose of further examination.

- Claims 33-35** depend on Claim 32 and, therefore, suffer the same deficiency as Claim 32.

Claim Rejections - 35 USC § 103

10. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

11. **Claims 1, 2, 4-7, 10, 11, 13-16, and 28-35** are rejected under 35 U.S.C. 103(a) as being unpatentable over US 6,289,506 (hereinafter “**Kwong**”) in view of US 6,412,109 (hereinafter “**Ghosh**”).

As per **Claim 1**, Kwong discloses:

- selecting a call to any native code method to be optimized within the virtual machine
(see Figure 7: 730; Column 8: 32-35, “... if the programmer decides to try to improve performance, then at step 730, he may select some of the Java program methods on the candidate list from step 720 for optimization.”);
- decompiling at least part of the selected native code method into an intermediate representation, wherein an intermediate representation includes a set of instruction code which is not in final executable form *(see Figure 7: 735; Column 8: 38-45, “... a user may decide to de-compile earlier native compiled code back to bytecode format. The de-compile process may be used for instance when a user determines that the native compiled code does not present the desired performance and the user wants to revert the native compiled code back to Java bytecode. A dynamic linked library (DLL) is created for the compiled native program methods at step 740.”);* [Examiner’s Remarks: Note that one of ordinary skill in the art would readily comprehend that code in the bytecode format is not in final executable form because it is typically JIT compiled to translate the bytecode to machine code for execution.]
- obtaining an intermediate representation associated with the application running on the virtual machine which interacts with the selected native code method *(see Figure 7: 705; Column 8: 23-25, “A programmer would first write a computer program in the Java*

programming language in step 705."); [Examiner's Note: The source code of the computer program is modified (intermediate representation) after an iteration of the optimization loop by incorporating the DLL for the native methods into the source code of the computer program.]

- integrating the intermediate representation for the selected native code method into the intermediate representation associated with the application running on the virtual machine to form an integrated intermediate representation (*see Figure 7: 705 and 740; Column 10: 8-10, "... the Java application now comprises of Lib.dll 1060, A.class 1010, and B.class 1020 and may be executed on a Java VM 1080.*"); [Examiner's Note: Figure 7 clearly illustrates that the DLL for the native methods (intermediate representation for the selected native code method) is incorporated into the modified source code of the computer program (intermediate representation associated with the application) and thus, the modified source code of the computer program now contains the DLL for the native methods (integrated intermediate representation).] and

- generating native code from the integrated intermediate representation, wherein the native code generation process optimizes interactions between the application running on the virtual machine and the selected native code method, wherein optimizing the interactions involves optimizing calls from the application to the selected native code method (*see Figure 7: 710, 715, and 720; Column 8: 46 and 47, "... a programmer may repeat these steps to further refine and optimize the program.*"; Column 9: 9-11, "Once the bytecodes are in the Java VM 840, they are interpreted by a Java interpreter 842 or turned into native machine code by the JIT compiler 844.").

However, Kwong does not disclose:

- using additional information from the integrated intermediate representation to reduce the number of indirect calls and indirect references associated with the calls from the application to the selected native code method.

Ghosh discloses:

- using additional information from an integrated intermediate representation to reduce the number of indirect calls and indirect references associated with calls from an application to a selected native code method (*see Column 4: 14-23, "As described above, an IR is a succinct and straightforward structuring of the control and data flow information of a program into a CFG and DFG. The CFG represents the information regarding the sequence and order of the execution of the bytecode instructions as blocks of code linked in the order of execution of the program. The basic blocks of code are sections of bytecode that are always performed as an uninterrupted group, and the links between the blocks of code are bytecode branch, conditional, or jump instructions."*).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Ghosh into the teaching of Kwong to modify Kwong's invention to include using additional information from the integrated intermediate representation to reduce the number of indirect calls and indirect references associated with the calls from the application to the selected native code method. The modification would be obvious because one of ordinary skill in the art would be motivated to generate bytecodes that would more efficiently perform the same operations (*see Ghosh – Column 2: 1-3*).

As per **Claim 2**, the rejection of **Claim 1** is incorporated; and Kwong further discloses:

- wherein selecting the call to any native code method involves selecting the call based upon at least one of: the execution frequency of the call (*see Column 4: 11-19, "An analysis tool may track the Java program methods entered and exited in memory, establish a relationship between parent and child methods called, record every called program method, and time spend in each method. In another embodiment, an analysis tool may keep track of the Java methods being loaded into memory along with active software executing on the system. A tuning tool may determine the most active classes and methods in a Java application and list possible candidates for native compilation."*); and the overhead involved in performing the call to any native code method as compared against the amount of work performed by the selected native code method.

As per **Claim 4**, the rejection of **Claim 1** is incorporated; and Kwong further discloses:

- wherein optimizing interactions between the application running on the virtual machine and the selected native code method involves optimizing callbacks by the selected native code method into the virtual machine (*see Column 7: 9-12, "In order to maintain the state of the Java VM 430 and make system calls, the compiled Java code 440 may make calls 450 into the Java VM 430."*).

As per **Claim 5**, the rejection of **Claim 4** is incorporated; however, Kwong and Ghosh do not disclose:

- wherein optimizing callbacks by the selected native code method into the virtual machine involves optimizing callbacks that access heap objects within the virtual machine.

Official Notice is taken that it is old and well-known within the computing art to allow callbacks to access heap objects within the virtual machine. Applicant has submitted in the “Background” section of the specification that JNI™ provides an interface through which native code can manipulate heap objects within the JVM™ in a platform-independent way (*see Page 2, Paragraph [0004]*). Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to modify Kwong’s invention to include wherein optimizing callbacks by the selected native code method into the virtual machine involves optimizing callbacks that access heap objects within the virtual machine. The modification would be obvious because one of ordinary skill in the art would be motivated to allow the implementation of a Java™ object to remain transparent to the native code.

As per **Claim 6**, the rejection of **Claim 4** is incorporated; and Kwong further discloses:

- wherein the virtual machine is a platform-independent virtual machine (*see Figure 2: 212*); and
- wherein integrating the intermediate representation for the selected native code method with the intermediate representation associated with the application running on the virtual machine involves integrating calls provided by an interface for accessing native code into the selected native code method (*see Column 5: 41-44, “A Java Native Interface (JNI) may exist with the Java VM 212. The Java Native Interface is a standard programming interface for writing Java native methods and embedding the Java VM into native applications.”; Column 10: 10-13, “When the method in A.class 1010 is native compiled, it needs to use the Java native interface 1070 to access the field b in class B 1020.”*).

As per **Claim 7**, the rejection of **Claim 1** is incorporated; and Kwong further discloses:

- wherein obtaining the intermediate representation associated with the application running on the virtual machine involves recompiling a corresponding portion of the application (see Column 8: 48-50, *"The process of monitoring and compiling bytecode/de-compiling native code may be repeated until the desired performance is obtained."*).

Claims 10, 11, and 13-16 are computer-readable storage device claims corresponding to the method claims above (Claims 1, 2, and 4-7) and, therefore, are rejected for the same reasons set forth in the rejections of Claims 1, 2, and 4-7.

As per **Claim 28**, Kwong discloses:

- deciding to optimize a callback by any native code method into the virtual machine (see Figure 7: 730; Column 7: 9-12, *"In order to maintain the state of the Java VM 430 and make system calls, the compiled Java code 440 may make calls 450 into the Java VM 430."*; Column 8: 32-35, *"... if the programmer decides to try to improve performance, then at step 730, he may select some of the Java program methods on the candidate list from step 720 for optimization."*);
- decompiling at least part of the selected native code method into an intermediate representation, wherein an intermediate representation includes a set of instruction code which is not in final executable form (see Figure 7: 735; Column 8: 38-45, *"... a user may decide to de-compile earlier native compiled code back to bytecode format. The de-compile process may be*

used for instance when a user determines that the native compiled code does not present the desired performance and the user wants to revert the native compiled code back to Java bytecode. A dynamic linked library (DLL) is created for the compiled native program methods at step 740."); [Examiner's Remarks: Note that one of ordinary skill in the art would readily comprehend that code in the bytecode format is not in final executable form because it is typically JIT compiled to translate the bytecode to machine code for execution.]

- obtaining an intermediate representation associated with the application running on the virtual machine which interacts with the selected native code method (*see Figure 7: 705; Column 8: 23-25, "A programmer would first write a computer program in the Java programming language in step 705."*); [Examiner's Note: The source code of the computer program is modified (intermediate representation) after an iteration of the optimization loop by incorporating the DLL for the native methods into the source code of the computer program.]

- integrating the intermediate representation for the selected native code method into the intermediate representation associated with the application running on the virtual machine to form an integrated intermediate representation (*see Figure 7: 705 and 740; Column 10: 8-10, "... the Java application now comprises of Lib.dll 1060, A.class 1010, and B.class 1020 and may be executed on a Java VM 1080."*); [Examiner's Note: Figure 7 clearly illustrates that the DLL for the native methods (intermediate representation for the selected native code method) is incorporated into the modified source code of the computer program (intermediate representation associated with the application) and thus, the modified source code of the computer program now contains the DLL for the native methods (integrated intermediate representation).] and

- generating native code from the integrated intermediate representation, wherein the native code generation process optimizes the callback by any native code method into the virtual machine, wherein optimizing the callback involves optimizing calls from the selected native code method to the application (*see Figure 7: 710, 715, and 720; Column 8: 46 and 47, "... a programmer may repeat these steps to further refine and optimize the program."*; Column 9: 9-11, *"Once the bytecodes are in the Java VM 840, they are interpreted by a Java interpreter 842 or turned into native machine code by the JIT compiler 844."*).

However, Kwong does not disclose:

- using additional information from the integrated intermediate representation to reduce the number of indirect calls and indirect references associated with the calls from the selected native code method to the application.

Ghosh discloses:

- using additional information from an integrated intermediate representation to reduce the number of indirect calls and indirect references associated with calls from a selected native code method to an application (*see Column 4: 14-23, "As described above, an IR is a succinct and straightforward structuring of the control and data flow information of a program into a CFG and DFG. The CFG represents the information regarding the sequence and order of the execution of the bytecode instructions as blocks of code linked in the order of execution of the program. The basic blocks of code are sections of bytecode that are always performed as an uninterrupted group, and the links between the blocks of code are bytecode branch, conditional, or jump instructions."*).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Ghosh into the teaching of Kwong to modify Kwong's invention to include using additional information from the integrated intermediate representation to reduce the number of indirect calls and indirect references associated with the calls from the selected native code method to the application. The modification would be obvious because one of ordinary skill in the art would be motivated to generate bytecodes that would more efficiently perform the same operations (*see Ghosh – Column 2: 1-3*).

As per **Claim 29**, the rejection of **Claim 28** is incorporated; and Kwong further discloses:

- wherein the native code generation process also optimizes calls to the selected native code method by the application (*see Column 8: 32-35, "... if the programmer decides to try to improve performance, then at step 730, he may select some of the Java program methods on the candidate list from step 720 for optimization."*).

As per **Claim 30**, the rejection of **Claim 28** is incorporated; however, Kwong and Ghosh do not disclose:

- wherein optimizing the callback by any native code method into the virtual machine involves optimizing a callback that accesses a heap object within the virtual machine.

Official Notice is taken that it is old and well-known within the computing art to allow callbacks to access heap objects within the virtual machine. Applicant has submitted in the "Background" section of the specification that JNI™ provides an interface through which native code can manipulate heap objects within the JVM™ in a platform-independent way (*see Page 2*,

Paragraph [0004]). Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to modify Kwong's invention to include wherein optimizing the callback by any native code method into the virtual machine involves optimizing a callback that accesses a heap object within the virtual machine. The modification would be obvious because one of ordinary skill in the art would be motivated to allow the implementation of a Java™ object to remain transparent to the native code.

As per **Claim 31**, the rejection of **Claim 28** is incorporated; and Kwong further discloses:

- wherein the virtual machine is a platform-independent virtual machine (*see Figure 2: 212*); and
- wherein integrating the intermediate representation for the selected native code method with the intermediate representation associated with the application running on the virtual machine involves integrating calls provided by an interface for accessing native code into the selected native code method (*see Column 5: 41-44, "A Java Native Interface (JNI) may exist with the Java VM 212. The Java Native Interface is a standard programming interface for writing Java native methods and embedding the Java VM into native applications."; Column 10: 10-13, "When the method in A.class 1010 is native compiled, it needs to use the Java native interface 1070 to access the field b in class B 1020."*).

Claims 32-35 are computer-readable storage device claims corresponding to the method claims above (Claims 28-31) and, therefore, are rejected for the same reasons set forth in the rejections of Claims 28-31.

12. **Claims 8 and 17** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Kwong** in view of **Ghosh** as applied to Claims 1 and 10 above, and further in view of US 5,491,821 (hereinafter “**Kilis**”).

As per **Claim 8**, the rejection of **Claim 1** is incorporated; however, Kwong and Ghosh do not disclose:

- wherein obtaining the intermediate representation associated the application running on the virtual machine involves accessing a previously generated intermediate representation associated with the application running on the virtual machine.

Kilis discloses:

- wherein obtaining an intermediate representation associated an application running on a virtual machine involves accessing a previously generated intermediate representation associated with the application running on the virtual machine (*see Column 2: 2-4, “If the selected changed facet affects the object itself, then the previous intermediate representation of the object is modified.”*).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Kilis into the teaching of Kwong to modify Kwong’s invention to include wherein obtaining the intermediate representation associated the application running on the virtual machine involves accessing a previously generated intermediate representation associated with the application running on the virtual machine. The modification would be obvious because one of ordinary skill in the art would be motivated to not

reprocess existing information and thus, reduce processing overhead (*see Kilis – Column 1: 40-43*).

Claim 17 is rejected for the same reason set forth in the rejection of Claim 8.

13. **Claims 9 and 18** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Kwong** in view of **Ghosh** as applied to Claims 1 and 10 above, and further in view of US 5,805,899 (hereinafter “**Evans**”).

As per **Claim 9**, the rejection of **Claim 1** is incorporated; and Kwong further discloses:

- determining a signature of the call to any native code method (*see Column 4: 11-19, “An analysis tool may track the Java program methods entered and exited in memory, establish a relationship between parent and child methods called, record every called program method, and time spend in each method. In another embodiment, an analysis tool may keep track of the Java methods being loaded into memory along with active software executing on the system. A tuning tool may determine the most active classes and methods in a Java application and list possible candidates for native compilation.”*).

However, Kwong and Ghosh do not disclose:

- determining a mapping from arguments of the call to corresponding locations in a native application binary interface (ABI).

Evans discloses:

- determining a mapping from arguments of a call to corresponding locations in a native application binary interface (ABI) (*see Column 7: 29-33, "Shared object 114 provides global symbols to which other objects, such as dynamic executable 120, can bind at runtime. These global symbols are specified in mapfile 130 and describe an Application Binary Interface (ABI) of shared object 114."*).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Evans into the teaching of Kwong to modify Kwong's invention to include wherein determining a mapping from arguments of the call to corresponding locations in a native application binary interface (ABI). The modification would be obvious because one of ordinary skill in the art would be motivated to describe the low-level interface between an application program and the operating system, its libraries, or components of the application program.

Claim 18 is rejected for the same reason set forth in the rejection of Claim 9.

Response to Arguments

14. Applicant's arguments filed on July 22, 2009 have been fully considered, but they are not persuasive.

In the Remarks, Applicant argues:

a) Applicant respectfully notes that Kwong does not disclose in any way generating an integrated intermediate representation. Kwong discloses a system which compiles a Java

application with Dynamic Link Libraries into an executable Java application, and monitors the performance of program code during program execution (see Kwong, columns 9-10 and abstract). Kwong explicitly states at the bottom of Column 9 that A.java 1005 and B.java 1015 "are compiled with a Java compiler 1030 to generate A.class 1010 and B.class 1020, respectively..." which "may then be executed on a Java VM 1080" (see Kwong, Column 9, line 65 - Column 10, line 3). Furthermore, a Dynamic Link Library (e.g., Lib.dll) is commonly known in the art to be a pre-compiled library of executable binary code. Therefore, the cited section of Column 10 in Kwong at most discloses combining multiple sources of executable binary code into a Java application, which is fundamentally distinct from generating an intermediate representation.

Applicant further notes that the cited compiler operations illustrated in Figure 7 of Kwong in no way hint at generating an integrated intermediate representation from native binary code and from original source code. At most, Figure 7 in Kwong illustrates the operations described in columns 9 and 10 of Kwong, which explicitly describes compiling source code files to generate executable files, and then combining these compiled executables with Dynamic Link Libraries to generate an executable Java application.

Examiner's response:

a) Examiner disagrees. Applicant's arguments are not persuasive for at least the following reasons:

First, without acquiescing to the Applicant's assertion that Kwong does not disclose in any way generating an integrated intermediate representation, as previously pointed out in the

Final Rejection (mailed on 11/24/2008) and further clarified hereinafter, the Examiner first submits that for at least the definiteness of the limitation of “intermediate representation” recited in the claims was analyzed in accordance with the level of ordinary skill in the art and in light of the specification. The specification provides an exemplary definition for “intermediate representation” as to “include any intermediate representation of code between the original source code and the final binary executable code for the application. For example, the intermediate representation can include, but is not limited to, modified source code, platform-independent byte codes, assembly code, an intermediate representation for computational operations used within a compiler, or binary code that is not in final executable form (emphasis added)” (*see paragraph [0028]*). Such description provided by the Applicant is, at most, an exemplary instance of the term rather than an *explicit and deliberate definition* for the term. Accordingly, the scope of “intermediate representation” is not limited to its ordinary and customary meaning as understood by those of ordinary skill in the art. Thus, as the claims are interpreted as broadly as their terms reasonably allow (*see* MPEP § 2111.01(I)), the interpretation of a broad limitation of “intermediate representation” as a DLL file and modified source code and the like by one of ordinary skill in the art is considered to be reasonable by its plain meaning and/or according to its exemplary definition as set forth in the specification.

Second, with respect to the Applicant’s assertion that Kwong does not disclose in any way generating an integrated intermediate representation, as previously pointed out in the Non-Final Rejection (mailed on 04/24/2009) and further clarified hereinafter, the Examiner respectfully submits that Kwong clearly discloses generating an integrated intermediate representation (*see Figure 7: 705 and 740; Column 10: 8-10, “... the Java application now*

comprises of Lib.dll 1060, A.class 1010, and B.class 1020 and may be executed on a Java VM 1080.”). Attention is drawn to Figure 7 of Kwong clearly illustrates that the DLL for the native methods (intermediate representation for the selected native code method) is incorporated into the modified source code of the computer program (intermediate representation associated with the application) and thus, the modified source code of the computer program now contains the DLL for the native methods (integrated intermediate representation).

Therefore, for at least the reasons set forth above, the rejections made under 35 U.S.C. § 103(a) with respect to Claims 1, 10, 28, and 32 are proper and therefore, maintained.

Conclusion

15. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

16. Any inquiry concerning this communication or earlier communications from the Examiner should be directed to Qing Chen whose telephone number is 571-270-1071. The Examiner can normally be reached on Monday through Thursday from 7:30 AM to 4:00 PM. The Examiner can also be reached on alternate Fridays.

If attempts to reach the Examiner by telephone are unsuccessful, the Examiner's supervisor, Wei Zhen, can be reached on 571-272-3708. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the TC 2100 Group receptionist whose telephone number is 571-272-2100.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

/Q. C./

Examiner, Art Unit 2191

/Wei Y Zhen/

Supervisory Patent Examiner, Art Unit 2191